# How to Write A Recipe?

## Automating Feature Engineering Using DriverlessAI


Ashrith Barthur

$H_2O$.ai


July 28, 2021

# Question

1. How many of us have built variables, features, transformers, or feature transformers?
2. What are they?

# Answer

1. Variables, features, transformers, feature transformers are refer to the same.
2. Each column in your data is considered a variable or a feature.
3. Each *new* column created is also referred to as a variable or a feature.
4. The process of creating a new variable, or a feature is called a transformation.
5. The code processing an *existing* column to a *new* column is called a *transformer*.

# Example Transformation

1. *height*- Variable
2. New variable after transformation *log2(height)*

# Question

1. How many of us are familiar with Custom Transformers in Driverless AI?
2. What are they?

# Answer

1. DriverlessAI already has a large, comprehensive set of transformers.
2. But there are always domains that require nuanced features.
3. And for this, DriverlessAI provides us to create custom transformers.
4. This is provided by provisioning an extension class *CustomTransformer*

# How Did We Build A Custom Transformer?

Driverless AI provides an extension.
This is a class ‘CustomTransformer‘

```
class ExampleLogTransformer(CustomTransformer):
```

# How Did We Build This?

The class has:

1. Parameters that need to be provided.
2. These parameters are specific to the type of feature recipe that you are building.
3. It also has four methods which primarily handle your feature engineering transformation.

# Parameters - Basic

```
class ExampleLogTransformer(CustomTransformer):
_regression = True
_binary = True
_multiclass = True
```

# Parameters - Advanced

```
class ExampleLogTransformer(CustomTransformer):
_regression = True
_binary = True
_multiclass = True
_numeric_output = True
_is_reproducible = True
_excluded_model_classes = ['tensorflow']
_modules_needed_by_name = ["custom_package==1.0.0"]
```

# Acceptance Method

```
class ExampleLogTransformer(CustomTransformer):
_regression = True
_binary = True
_multiclass = True
_numeric_output = True
_is_reproducible = True
_excluded_model_classes = ['tensorflow']
_modules_needed_by_name = ["custom_package==1.0.0"]

@staticmethod
def do_acceptance_test():
return True
```

# Input Data

```
...
@staticmethod
def do_acceptance_test():
return True

@staticmethod
def get_default_properties():
return dict(col_type = "numeric", min_cols = 1, max_cols = 1,
relative_importance = 1)
```

# Input Data Types

```
a. "all"         - all column types
b. "any"         - any column types
c. "numeric"     - numeric int/float column
d. "categorical" - string/int/float column considered a categorical for
feature engineering
e. "numcat"      - allow both numeric or categorical
f. "datetime"    - string or int column with raw datetime such as
'%Y/%m/%d %H:%M:%S' or '%Y%m%d%H%M'
```

# Input Data Types

```
g. "date"          - string or int column with raw date such as
'%Y/%m/%d' or '%Y%m%d'
h. "text"          - string column containing text
(and hence not treated as categorical)
i. "time_column" - the time column specified at the start of
the experiment (unmodified)
```

# Feature Importance

1. Feature Importance (FI) is a probability of feature being used.
2. All features have a standard FI of 1, but they can be changed depending on their importance.
3. Increasing the FI beyond 1 will increase the feature's probability of being used in the model.

# Number of Columns

1. min_cols and max_cols set the minimum and the maximum columns the feature can work with.
2. It also tells DAI to only pass the given max/min number of columns.

# Fit Function

```
@staticmethod
def get_default_properties():
return dict(col_type = "numeric", min_cols = 1, max_cols = 1,
relative_importance = 1)

def fit_transform(self, X: dt.Frame, y: np.array = None):
X_pandas = X.to_pandas()
X_p_log = np.log2(X_pandas)
return X_p_log
```

# Transform Function

```
def fit_transform(self, X: dt.Frame, y: np.array = None):
X_pandas = X.to_pandas()
X_p_log = np.log2(X_pandas)
return X_p_log

def transform(self, X: dt.Frame):
X_pandas = X.to_pandas()
X_p_log = np.log2(X_pandas)
return X_p_log
```

# Library

```
from h2oaicore.systemutils import segfault, loggerinfo, main_logger
from h2oaicore.transformer_utils import CustomTransformer
import datatable as dt
import numpy as np
import pandas as pd
import logging
```

# DEMO

# Advantages

1. Feature engineering process standardised by:
    1.1 preset parameters
    1.2 preset methods
2. Effort minimisation leads to minimisation in time spent.
3. Build only once - Feature engineering is carried over from training/testing to production.
4. DAI automatically, runs multiple models on various sets of features to get the best model.
5. All the requirements are handled internally by DAI.

# References

**How to build a recipe**

https://github.com/ashrith/how_to_write_a_recipe

# Thanks & Questions